Automatic Proof Generation in Kleene Algebra with Tests

James Worthington

January 27, 2007

Abstract

Kleene algebra (KA) is the algebra of regular events. Familiar examples of Kleene algebras include regular sets, relational algebras, and trace algebras. A Kleene algebra with tests (KAT) is a Kleene algebra with an embedded Boolean subalgebra. The addition of tests allows one to encode while programs as KAT terms, thus the equational theory of KAT can express (propositional) program equivalence. More complicated statements about programs can be expressed in the Hoare theory of KAT, which suffices to encode Propositional Hoare Logic.

In this paper, we prove the following. First, there is a $PSPACE$ transducer which takes equations of Kleene algebra as input and outputs Hilbert-style proofs of them in an equational implication calculus. Second, we give a feasible reduction from the equational theory of KAT to the equational theory of KA. Combined with the fact that the Hoare theory of KAT reduces efficiently to the equational theory of KAT, this yields an algorithm capable of generating proofs of a large class of statements about programs.

1 Introduction

The class of Kleene algebras is defined by equations and equational implications over the signature $\{0, 1, +, \cdot, ^* \}$. Some well-known examples of Kleene algebras include relational algebras, trace algebras, and sets of regular languages (see [1] for more examples and applications). In fact, the set of regular languages over an alphabet Σ is the free Kleene algebra on Σ. That is, given two KA terms α and β, $\alpha = \beta$ modulo the axioms of Kleene algebra if and only if α and β denote the same regular set [4]. A Kleene algebra with tests is a Kleene algebra with an embedded Boolean subalgebra (the complementation function is only defined on Boolean terms).

Adding tests allows the encoding of while programs as KAT terms. As a result, the equational theory of KAT suffices to express (propositional) equivalence of while programs. Moreover, Propositional Hoare Logic can be encoded in the Hoare theory of KAT (equational implications of the form $r = 0 \to p = q$), and furthermore the Hoare theory of KAT reduces efficiently to the equational theory of KAT. See [6], [8], and [9] for details.

In [5], it is shown that the equational theory of KAT reduces to the equational theory of KA. Unfortunately, the reduction used can increase the size of the terms involved exponentially. One of our main results is an alternate reduction, which increases the size of the terms by only a polynomial amount. Combining all of these reductions shows that the equational theory of KA can be used to express interesting properties of programs succinctly.

Our second result is that the production of proofs of KA equations can be automated: there is a $PSPACE$ transducer which takes as input equations of Kleene algebra and outputs Hilbert-style proofs of them in an equational implication calculus. This construction is a significant simplification of the original completeness result of [4]. The proofs are exponentially long in the worst case, but this is the best that one could expect, unless $PSPACE = NP$. Deciding the equational theory of KA is a $PSPACE$ complete problem [13], so the existence of polynomially long proofs of all equivalences would imply $PSPACE = NP$.

Until now, there has been no approach to the equational theory of KA that is both completely automated and produces formal proofs. The well-known algorithm of Stockmeyer and Meyer to decide whether two finite automata accept the same language [13] can obviously be performed by a machine, but the downside of this method is that the output is just one bit, and therefore not efficiently verifiable. Furthermore, this method can not be used for applications which require the production of a formal proof, such as Proof-Carrying Code [10] [11].

Human-aided proof generation using a proof assistant is another possible approach to the equational theory of KA. Unfortunately, such provers are by definition not automated, which poses a problem for applications. Our result shows that it is possible to have all of the advantages and none of the drawbacks: independently verifiable proof artifacts can be produced by a machine.

This paper is organized as follows. In section 2, we provide the relevant definitions and show how to encode finite automata as Kleene algebra terms. In section 3, we prove some useful theorems of Kleene algebra used for reasoning about automata. In section 4, we give a $PSPACE$ transducer which takes an equation of KA as input and outputs a proof of it. In section 5, we give a feasible reduction from the equational theory of KAT to the equational theory of KA. Finally, in section 6, we show that the Hoare theory of KA(T) can be efficiently reduced to the equational theory of KA(T).

2 Background

In this section, we describe our proof system and recall some useful facts about KA(T). The axiomatization of Kleene algebra, results about matrices, and the encoding of automata as KA terms are from [4]. The definition of KAT is from [6].

2.1 Equational Logic

By "proof", we mean a sequent in the equational implication calculus. Let $\alpha, \beta, \gamma, \delta$ be terms in the language of Kleene algebra. The equational axioms are:

$$\begin{aligned}
&\alpha = \alpha \\
&\alpha = \beta \to \beta = \alpha \\
&\alpha = \beta \to \beta = \gamma \to \alpha = \gamma \\
&\alpha = \beta \to \gamma = \delta \to \alpha + \gamma = \beta + \delta \\
&\alpha = \beta \to \gamma = \delta \to \alpha \cdot \gamma = \beta \cdot \delta \\
&\alpha = \beta \to \alpha^* = \beta^*.
\end{aligned}$$

We consider these Horn formulas to be implicitly universally quantified.

Let Φ be a sequence of equations or equational implications, e an equation, ϕ a Horn formula, and ψ an equational axiom or an axiom of KA (given below). Let σ be a substitution of terms for variables. The rules of inference are:

$$\vdash \sigma(\psi) \qquad e \vdash e \qquad \frac{\Phi \vdash \phi}{\Phi, e \vdash \phi} \qquad \frac{\Phi, e \vdash \phi}{\Phi \vdash e \to \phi} \qquad \frac{\Phi \vdash e \qquad \Phi \vdash e \to \phi}{\Phi \vdash \phi},$$

and the structural rules which allow us to treat a sequence of formulas as a set of formulas. For a proof that this is a complete deductive system, see [12]. We also allow "substitution of equals for equals". For example, from $a = b$, conclude $c(a + 1) = c(b + 1)$ in one step.

2.2 Kleene Algebra

We now state the axioms of Kleene algebra. The first are the idempotent semiring axioms. Note that we abbreviate $\alpha \cdot \beta$ as $\alpha\beta$.

1. $(a + b) + c = a + (b + c)$
2. $a + b = b + a$
3. $a + 0 = a$
4. $a + a = a$
5. $(ab)c = a(bc)$
6. $1a = a1 = a$
7. $a(b + c) = ab + ac$
8. $(a + b)c = ac + bc$
9. $0a = 0a = 0$

In any idempotent semiring, addition can be used to define a partial order:

$$x \leq y \Leftrightarrow x + y = y.$$

For brevity, we add the symbol $\leq$ to the language.

There are four axioms involving *. The equational axioms are:

10. $1 + xx^* = x^*$
11. $1 + x^*x = x^*$

There are also two equational implications:

12. $b + ax \leq x \rightarrow a^*b \leq x$
13. $b + xa \leq x \rightarrow ba^* \leq x$

The equational implications guarantee unique least solutions to the linear inequalities

$$b + aX \leq X$$

$$b + Xa \leq X$$

in the presence of the other axioms.

2.3 Kleene Algebra with Tests

A Kleene algebra with tests is a Kleene algebra with an embedded Boolean subalgebra; Boolean terms are called tests. Formally, a Kleene algebra with tests is a two-sorted structure $(K, B, +, \cdot, ^*, ^-, 0, 1)$ such that $(K, +, \cdot, ^*, 0, 1)$ is a Kleene algebra and $(B, +, \cdot, ^-, 0, 1)$ is a Boolean algebra. Note that complementation is only defined on tests.

We use the following axiomatization of Boolean algebra. Let b, c, d be Boolean terms.

1. KA axioms 1 - 9
2. $\overline{0} = 1;\ \overline{1} = 0$
3. $b + 1 = 1$
4. $b\overline{b} = \overline{b}b = 0$

5. $\overline{\overline{b}} = b$

6. $bb = b$

7. $\overline{b + c} = \overline{b}\overline{c}$; $\overline{bc} = \overline{b} + \overline{c}$

8. $bc = cb$

9. $b + cd = (b + c)(b + d)$

Any Boolean term b satisfies $b \leq 1$. Since $1^* = 1$ and * is monotonic, the KA axioms imply $b^* = 1$. Note that any Kleene algebra can be viewed as a KAT with $\{0, 1\}$ as the two-element Boolean subalgebra.

2.4 Matrices and Automata

The Kleene algebra axioms imply that the set of $n \times n$ matrices over a KA also forms a KA. Addition and multiplication of matrices are defined in the usual way, 0 is interpreted as the $n \times n$ zero matrix, and 1 as I_n. Equality and the partial order $\leq$ are defined componentwise. To define the star of an $n \times n$ matrix, we first define the star of a 2×2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^* = \begin{bmatrix} (a + bd^*c)^* & (a + bd^*c)bd^* \\ (d + ca^*b)ca^* & (d + ca^*b)^* \end{bmatrix}.$$

We then extend this definition to arbitrary square matrices inductively. Given a square matrix E, partition E into four submatrices

$$E = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

such that A and D are square. By induction, A^* and D^* exist. Let $F = A + BD^*C$. Then

$$E^* = \left[\begin{array}{c|c} F^* & F^*BD^* \\ \hline D^*CF^* & D^* + D^*CF^*BD^* \end{array}\right].$$

It is a consequence of the KA axioms that any partition may be chosen to compute E^*.

In [3], it is shown that the set of $n \times n$ matrices over a Kleene algebra with tests is a Kleene algebra with tests. The Boolean subalgebra is the set of matrices with Boolean terms on the diagonal and all other entries equal to 0.

At several points in the proof below, we will have to reason about non-square matrices. We would like to know whether the theorems of Kleene algebra hold when the primitive letters are interpreted as matrices of arbitrary dimension and the function symbols are treated polymorphically. In general, the answer is no. However, there is a large class of theorems for which this does hold, and they suffice for our purposes. See [7] for a thorough treatment of this issue.

To make matters precise, we assume that each term is equipped with a type annotation of the form $s \to t$. We allow s and t to range over the positive natural numbers, and interpret s as the row dimension of a matrix and t as the column dimension. We assume that there are variables of all possible types, and that a verifier could determine the type of a term by an elementary inductive procedure. Addition and multiplication must respect the types, and we require that * is only applied to square matrices.

We must revise our axioms to take types into account. Since the $n \times n$ matrices over a KA form a KA, we allow all KA axioms where all of the terms appearing have the same square type. The typed multiplicative identity $1_{n \to n}$ is I_n, and the typed additive identity $0_{n \to n}$ is $Z_{n \times n}$, the $n \times n$ matrix of zeroes. To handle non-square matrices, we define $0_{s \to t}$ to be $Z_{s \times t}$, and of the Kleene algebra axioms 1-9 above, we extend all but (6) to their well-typed versions. Note that a regular

expression can be viewed as a KA term of type $1 \to 1$. Note that to prove two matrices equivalent, it suffices to prove that all corresponding entries are equivalent.

We now show how to use matrices over a KA to encode finite automata.

Definition 1. *An* automaton *over a Kleene algebra K is a triple (u, A, v) where u and v are n-dimensional vectors with entries from $\{0, 1\}$ and A is an $n \times n$ matrix over K. The vector u encodes the start states of the automaton and is called the* start vector. *The vector v encodes the accept states of the automaton and is called the* accept vector. *The matrix A is called the* transition matrix. *The language accepted by (u, A, v) is $u^{\mathrm{T}} A^* v$. The* size *of (u, A, v) is the number of states, i.e., if A is an $n \times n$ matrix, then the size of (u, A, v) is n.*

This definition is a bit general for the purposes at hand. Given an alphabet Σ, let $\mathcal{F}_\Sigma$ be the free Kleene algebra on generators Σ. Over $\mathcal{F}_\Sigma$, the definition of an automaton given above is essentially the same as the classical definition of a finite automaton. In the sequel, all automata are over some $\mathcal{F}_\Sigma$. Furthermore, most of the automata we consider have uncomplicated transition matrices.

Definition 2. *Let (u, A, v) be an automaton over $\mathcal{F}_\Sigma$. The automaton (u, A, v) is* simple *if A can be expressed as a sum*

$$A = J + \sum_{a \in \Sigma} a \cdot A_a$$

where J and each A_a is a 0-1 matrix.
The automaton (u, A, v) is ϵ-free *if J is the zero matrix.*
The automaton (u, A, v) is deterministic *if it is simple, ϵ-free, and u and all rows of each A_a have exactly one 1.*

Given an automaton (u, A, v), we denote the transition relation encoded by A as δ_A, and the extended transition relation defined on (states,words) as $\hat{\delta}_A$. Given an $a \in \Sigma$, we denote the restriction of δ_A to only a-transitions by δ_A^a. For transition matrices A, B, C, we denote the underlying state sets of the automata by $\mathcal{A}, \mathcal{B}, \mathcal{C}$. We now state the theorems of KA which we will use to reason about automata.

3 Useful Theorems of KA

The completeness result of [4] uses the fact that automata can be encoded as KA terms. To simplify proofs, we add several theorems of Kleene algebra involving automata to our list of allowable rules of inference. For each theorem we add, it will be clear that the hypotheses of the theorem are easy to check, so proofs constructed using these new rules of inference are verifiable in polynomial time. Several of the theorems about automata are based on the following theorems of Kleene algebra:

$$(x + y)^* = x^*(yx^*)^*$$
$$ay = yb \to a^*y = yb^*$$
$$x(yx)^* = (xy)^*x.$$

These are known as the *denesting, bisimulation*, and *sliding* rules, respectively. See [4] for a proof that these rules are consequences of the KA axioms.

Our first three theorems are used in the transformation of a regular expression to an equivalent automaton, i.e., instances of Kleene's theorem. Given a regular expression α, let $R(\alpha)$ be the regular set denoted by α.

3.1 The Union Lemma

Let (u, A, v) and (s, B, t) be automata and γ, δ be regular expressions. Suppose $u^{\mathrm{T}} A^* v = \gamma$ and $s^{\mathrm{T}} B^* t = \delta$. Combinatorially, we can construct an automaton accepting $\gamma + \delta$ by taking the disjoint union of (u, A, v) and (s, B, t). The *union* lemma,

$$\frac{\Phi \vdash u^{\mathrm{T}} A^* v = \gamma \qquad \Phi \vdash s^{\mathrm{T}} B^* t = \delta}{\Phi \vdash \left[\begin{array}{c} u \\ \hline s \end{array}\right]^{\mathrm{T}} \left[\begin{array}{c|c} A & 0 \\ \hline 0 & B \end{array}\right]^* \left[\begin{array}{c} v \\ \hline t \end{array}\right] = \gamma + \delta},$$

shows that this construction can be encoded algebraically.

The proof is straightforward; it relies on the fact that any partition can be used to compute the star of a matrix.

$$\left[\begin{array}{c|c} u & s \end{array}\right] \left[\begin{array}{c|c} A & 0 \\ \hline 0 & B \end{array}\right]^* \left[\begin{array}{c} v \\ \hline t \end{array}\right] = \left[\begin{array}{c|c} u & s \end{array}\right] \left[\begin{array}{c|c} A^* & 0 \\ \hline 0 & B^* \end{array}\right] \left[\begin{array}{c} v \\ \hline t \end{array}\right] = u^{\mathrm{T}} A^* v + s^{\mathrm{T}} B^* t = \gamma + \delta.$$

3.2 The Concatenation Lemma

Given two automata, (u, A, v) and (s, B, t), accepting γ and δ, respectively, one can combinatorially construct an automaton accepting $\gamma\delta$ by adding ϵ-transitions from the accept states of (u, A, v) to the start states of (s, B, t). The construction can also be modeled algebraically, yielding the *concatenation* lemma:

$$\frac{\Phi \vdash u^{\mathrm{T}} A^* v = \gamma \qquad \Phi \vdash s^{\mathrm{T}} B^* t = \delta}{\Phi \vdash \left[\begin{array}{c} u \\ \hline 0 \end{array}\right]^{\mathrm{T}} \left[\begin{array}{c|c} A & vs^{\mathrm{T}} \\ \hline 0 & B \end{array}\right]^* \left[\begin{array}{c} 0 \\ \hline t \end{array}\right] = \gamma\delta}.$$

The proof of the concatenation lemma is similar to that of the union lemma.

$$\left[\begin{array}{c|c} u & 0 \end{array}\right] \left[\begin{array}{c|c} A & vs^{\mathrm{T}} \\ \hline 0 & B \end{array}\right]^* \left[\begin{array}{c} 0 \\ \hline t \end{array}\right] = \left[\begin{array}{c|c} u & 0 \end{array}\right] \left[\begin{array}{c|c} A^* & A^* v s^{\mathrm{T}} B^* \\ \hline 0 & B^* \end{array}\right] \left[\begin{array}{c} 0 \\ \hline t \end{array}\right] = u^{\mathrm{T}} A^* v s^{\mathrm{T}} B^* t = \gamma\delta.$$

3.3 The Asterate Lemma

Let (u, A, v) be an automaton accepting $R(\gamma)$. Combinatorially, one can construct an automaton accepting $R(\gamma^*)$ by first adding ϵ-transitions from the accept states of (u, A, v) to the start states, and then adding an additional state to accept the empty word. To encode this construction algebraically, we first argue that the automaton

$$(u, A + vu^{\mathrm{T}}, v)$$

accepts $\gamma\gamma^*$. The proof relies on the denesting and sliding rules of Kleene algebra:

$$u^{\mathrm{T}}(A + vu^{\mathrm{T}})^* v = u^{\mathrm{T}} A^* (vu^{\mathrm{T}} A^*)^* v = u^{\mathrm{T}} A^* v (u^{\mathrm{T}} A^* v)^*.$$

We then add this automaton to the trivial one-state automaton accepting the empty word. This justifies the *asterate* lemma:

$$\frac{\Phi \vdash u^{\mathrm{T}} A^* v = \gamma}{\Phi \vdash \left[\begin{array}{c} 1 \\ \hline u \end{array}\right]^{\mathrm{T}} \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & A + vu^{\mathrm{T}} \end{array}\right]^* \left[\begin{array}{c} 1 \\ \hline v \end{array}\right] = \gamma^*}.$$

3.4 The ϵ-closure Lemma

The ϵ-closure lemma is based on the denesting rule. Let (u, A, v) and (s, B, v) be automata of size n, and let J be an $n \times n$ matrix. Suppose that the following equations hold:

$$A = J + A'$$
$$B = A'J^*$$
$$s^{\mathrm{T}} = u^{\mathrm{T}}J^*.$$

Then we can use the denesting rule to prove the equivalence of (u, A, v) and (s, A', v):

$$s^{\mathrm{T}}B^*v = u^{\mathrm{T}}J^*(A'J^*)^*v = u^{\mathrm{T}}(J + A')^*v = u^{\mathrm{T}}A^*v.$$

We allow the following rule of inference, called the *ϵ-closure* lemma:

$$\frac{\Phi \vdash A = J + A' \qquad \Phi \vdash B = A'J^* \qquad \Phi \vdash s^{\mathrm{T}} = u^{\mathrm{T}}J^*}{\Phi \vdash u^{\mathrm{T}}A^*v = s^{\mathrm{T}}B^*v}.$$

In our applications, J is a 0-1 matrix, so $u^{\mathrm{T}}J^*$ is a 0-1 vector and B is ϵ-free.

3.5 The Bisimulation Lemma

The bisimulation lemma plays a crucial role in our proofs because it allows us to reason about the language accepted by an automaton without having to compute the star of its transition matrix. In [4], it is shown that the bisimulation rule holds for a an $n \times n$ matrix, b an $m \times m$ matrix, and y an $n \times m$ matrix over a KA. In our applications, a and b are transition matrices, and y is a 0-1 matrix encoding a relation between states of the automata.

Let (u, A, v) and (s, B, t) be automata, and $R \subseteq \mathcal{B} \times \mathcal{A}$. Let Y be the realization of R as a 0-1 matrix, and suppose

$$YA = BY$$
$$s^{\mathrm{T}}Y = u^{\mathrm{T}}$$
$$Yv = t.$$

Then the bisimulation rule can be used to prove that (u, A, v) and (s, B, t) are equivalent:

$$u^{\mathrm{T}}A^*v = s^{\mathrm{T}}YA^*v = s^{\mathrm{T}}B^*Yv = s^{\mathrm{T}}B^*t.$$

We combine the hypothesis of the bisimulation rule with the above conditions on the start and accept vectors to define the *bisimulation lemma*:

$$\frac{\Phi \vdash YA = BY \qquad \Phi \vdash Yv = t \qquad \Phi \vdash s^{\mathrm{T}}Y = u^{\mathrm{T}}}{\Phi \vdash u^{\mathrm{T}}A^*v = s^{\mathrm{T}}B^*t}.$$

We call such a matrix Y a *bisimulation matrix*. We call (s, B, t) the *source* automaton, and (u, A, v) the *target* automaton. It is not hard show that direction matters, i.e., there exist automata (u, A, v) and (s, B, t) which can be proven equivalent using the bisimulation lemma with (s, B, t) as the target and (u, A, v) as the source, but not vice versa.

Moreover, given two equivalent automata, it is not necessarily the case that the bisimulation lemma can be used to prove their equivalence. Here is an example:

$$\left(\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0 & a & a\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}\right), \left(\begin{bmatrix}0\\0\\1\end{bmatrix}, \begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & a & a\end{bmatrix}, \begin{bmatrix}1\\1\\0\end{bmatrix}\right).$$

Each automaton accepts the language $\{a\}$, but a simple calculation shows that attempting to solve either of the systems

$$\begin{array}{cc} s^{\mathrm{T}}Y = u^{\mathrm{T}} & u^{\mathrm{T}}Y = s^{\mathrm{T}} \\ Yv = t & Yt = v \\ YA = BY & YB = AY \end{array}$$

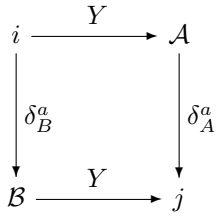
for Y yields an inconsistency. This is not surprising; nfa equivalence is a $PSPACE$ complete problem. If the bisimulation lemma could be used to prove any equivalence between automata, then $PSPACE = NP$, because a machine could just guess a bisimulation matrix between two given automata and then verify that the hypotheses of the bisimulation lemma hold. Moreover, it is not even the case that any two equivalent deterministic finite automata can be proven equivalent using the bisimulation lemma, even though this restriction would avoid $NP = PSPACE$. Consider the deterministic automata

$$\left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & a & 0 \\ 0 & 0 & a \\ a & 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right), \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & a \\ a & 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right).$$

Both automata accept the language a^*, but neither system of equations has a solution.

This means that before we can use the bisimulation lemma to prove the equivalence of two automata, we must ensure that the lemma applies. We will deal with the equations relating the bisimulation matrix, start vectors, and accept vectors on a case by case basis. In [4], it is noted that the equation $YA = BY$ means that the actions of the two automata commute in the appropriate spaces. We express this consideration as a diagram, and use the diagram to convince ourselves of the truth of $YA = BY$. Constructing the proof is another matter, we just want to check that we haven't given bad input to the proof generator.

Lemma 1. *Let (u, A, v) and (s, B, t) be simple, ϵ-free automata, and Y a relation from the states of (s, B, t) to the states of (u, A, v). Suppose that for each $a \in \Sigma$, $i \in \mathcal{B}$, and $j \in \mathcal{A}$, the diagram*



commutes, i.e., there is a path from i to j above the diagonal if and only if there is a path below the diagonal. Then $YA = BY$.

Proof. It suffices to show that for all i, j, $(YA)_{ij} = (BY)_{ij}$. We first unwind the definitions of YA and BY:

- $(YA)_{ij}$ =(i^{th} row of Y) $\cdot$ (j^{th} column of A), a sum consisting of the labels of the incoming transitions to state j in (u, A, v) from states of (u, A, v) related to state i of (s, B, t).
- $(BY)_{ij}$ =(i^{th} row of B) $\cdot$ (j^{th} column of Y), a sum consisting of the labels of the outgoing transitions of state i of (s, B, t) to states of (s, B, t) related to state j of (u, A, v).

The commutativity condition implies that for each $a \in \Sigma$, $a \leq (YA)_{ij} \leftrightarrow a \leq (BY)_{ij}$. Since (u, A, v) and (s, B, t) are simple, $YA = BY$. Note that because $a + a = a$, it does not matter how many times a appears in $(YA)_{ij}$ or $(BY)_{ij}$, only whether a appears. □

4 Proving KA Equations

In this section, we prove one of our main theorems. Given a regular expression α, let $|\alpha|$ be the number of symbols in α.

Theorem 1. *Let α and β be two equivalent regular expressions over an alphabet Σ. A proof that $\alpha = \beta$ can be produced by a transducer using only polynomially many (in $|\alpha| + |\beta|$) worktape cells.*

Respecting the space bound is nontrivial; we require several terms of exponential size, some of which are constructed from terms which are themselves exponentially large. In addition, we must not only construct these large terms, but also construct proofs of statements involving them. To ensure that everything can be done without violating the space bound, we divide the construction of the proof into stages. For each stage, we show that both the terms and the proofs required at that stage can be constructed in $PSPACE$. The stages:

1. Construct an nfa accepting α, an nfa accepting β, and proofs thereof.
2. For each nfa, construct an equivalent ϵ-free nfa, and an equivalence proof.
3. For each ϵ-free nfa, construct an equivalent dfa, and an equivalence proof.
4. Construct the minimal dfa equivalent to the dfa accepting α, and an equivalence proof.
5. Construct a proof of the equivalence of the dfa accepting β and the minimal dfa accepting α.

Stages 2 through 5 require one or more terms from previous stages. We treat each stage independently, and show that there are term-generating transducers to generate the required terms, and proof-generating transducers to generate the required proofs. To combine all of the stages, we use the following fact about the composition of space-bounded transducers.

Lemma 2. *Suppose f(x) can be computed by a PSPACE transducer F, and g(x) can be computed by a PLSPACE transducer G (a transducer using polylog many worktape cells in the size of its input). Then g(f(x)) can be computed by a PSPACE transducer.*

Proof. Note that $f(x)$ might be exponential in $|x|$, so there is not necessarily enough space to write down $f(x)$ in its entirety. Rather, a $PSPACE$ transducer H computing $g(f(x))$ computes $f(x)$ on a demand-driven basis. On input x, H begins by running G. Whenever a bit of $f(x)$ is needed, H saves the current state of M and begins running F on input x, disregarding the output of F until the required bit of $f(x)$ is produced. It then resumes running M, supplying the requested bit of $f(x)$. The transducer H needs polynomially many worktape cells to run F, polynomially many cells to count up to the length of $f(x)$, and polynomially many cells for G's worktape, since G needs at most $O((\log |f(x)|)^d) \leq O(|x|^m)$ for some m. □

We denote by P the machine with takes as input a pair of equivalent regular expressions (α, β) and outputs an equivalence proof. Each of the stages below is essentially a description of part of P.

4.1 KA Term to Automaton

We first show that the inductive construction used in the proof of Kleene's theorem can be performed by a $PSPACE$ machine. Given a regular expression γ, the machine must construct an automaton (u, A, v) accepting γ, and a proof that $u^{\mathrm{T}} A^* v = \gamma$. Note that at this stage, the proof-generating transducer and the term-generating transducer coincide.

Given $a \in \Sigma$, the following automaton accepts the language $\{a\}$:

$$\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right).$$

There are also one-state automata for $\emptyset$ and ϵ: $([0],[0],[0])$ and $([1],[1],[1])$, respectively. We assume that for every $a \in \Sigma$, the machine has a proof that

$$a = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & a \\ 0 & 0 \end{bmatrix}^* \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

stored in its finite control. We also assume that the machine can output proofs of the equations

$$0 = 00^*0$$

$$1 = 11^*1.$$

For the inductive step, the machine can work its way up the syntax tree of γ, constructing automata as dictated by the union, concatenation, and asterate lemmas. At each step, it outputs the appropriate equation, i.e., the conclusion of one of the three lemmas. When finished, the machine will have constructed an automaton accepting α and also will have printed a proof of this fact on the output tape. All of the terms appearing in the proof are polynomial in the size of γ and straightforward to construct, so a $PSPACE$ transducer could generate the proof.

The main transducer, P, performs the above procedure to generate automata (u_1, A, v_1) and (u_2, B, v_2) accepting α and β, respectively, and outputs proofs thereof. It also stores these automata on its worktape.

4.2 Automaton to ϵ-free Automaton

We now show that there is a term-generating transducer which takes a simple automaton (u, A, v) as input and constructs from it an equivalent ϵ-free automaton, and that there is a proof-generating transducer which takes as input the pair (automaton, equivalent ϵ-free automaton) and outputs a proof of the equivalence.

Constructing the ϵ-free automaton, (s, F, v), is easy. Since (u, A, v) is simple,

$$A = J + \sum_{a \in \Sigma} a \cdot A_a.$$

The transducer can easily compute J from (u, A, v) and then compute J^*, which is just the reflexive transitive closure of the relation denoted by J. It can also compute

$$A' = \sum_{a \in \Sigma} a \cdot A_a.$$

Then

$$s^{\mathrm{T}} = u^{\mathrm{T}} J^*$$

$$F = A' J^*.$$

It is easy to see that both s and F can be constructed in $PSPACE$.

To prove equivalence, the proving transducer uses the ϵ-elimination lemma. It must prove the following hypotheses:

$$A = J + A'$$

$$B = A' J^*$$

$$s^{\mathrm{T}} = u^{\mathrm{T}} J^*$$

all of which are easily proven in $PSPACE$. The machine must also prove that the term J^* actually is the star of J. Note that J is an $n \times n$ 0-1 matrix. First, the machine proves

$$1 + J(1 + J + J^2 + \cdots + J^n) \leq (1 + J + J^2 + \cdots + J^n)$$

by direct computation. This inequality is true; if the i, j entry of JJ^n is 1, then there is a path of length $n+1$ from i to j if we view J as the adjacency matrix of a graph. Since J has only n vertices, this path must repeat at least one vertex, and so there will be a 1 in the i, j position of J^k for some $k < n + 1$. By KA axiom 12 and some algebraic simplification,

$$J^* \leq 1 + J + J^2 + \cdots + J^n.$$

Next, the machine generates a proof that for any x,

$$1 + x + x^2 + \cdots + x^n \leq x^*.$$

This inequality is an easy consequence of the KA axioms. Substituting J for X and combining these two inequalities yields

$$1 + J + J^2 + \cdots + J^n = J^*.$$

The transducer P uses the term-generating procedure to construct (s_1, F_A, v_1), an ϵ-free automaton accepting α, and (s_2, F_B, v_2), an ϵ-free automaton accepting β. It stores both of these automata on its worktape. It then applies the proof-generating procedure to output proofs that $u_1^{\mathrm{T}} A^* v_1 = s_1^{\mathrm{T}} F_A^* v_1$ and $u_2^{\mathrm{T}} B^* v_2 = s_2^{\mathrm{T}} F_B^* v_2$. Finally, P outputs $\alpha = s_1^{\mathrm{T}} F_A^* v_1$ and $\beta = s_2^{\mathrm{T}} F_B^* v_2$, both of which follow by transitivity.

4.3 ϵ-free Automaton to Deterministic Automaton

It must now be shown that there is a $PSPACE$ transducer which takes in (s, F, v), an ϵ-free automaton, and outputs (p, D, t), an equivalent deterministic automaton. We must also show that there is a proof-generating transducer to prove $s^{\mathrm{T}} F^* v = p^{\mathrm{T}} D^* t$. The proof-generating transducer requires one additional term: the bisimulation matrix between the two automata. Let $|(s, F, v)| = n$.

To generate (p, D, t), the term-generating machine performs the standard subset construction on (s, F, v), with the added condition that it tests each subset for accessibility before granting it state status. We must show that this test can be performed in $PSPACE$.

Lemma 3. *Let (s, F, v) be an ϵ-free automaton with n states. It is decidable in $O(n^2)$ space whether C, a set of states of (s, F, v), is accessible when considered as a state in the deterministic automaton obtained from (s, F, v) by the subset construction.*

Proof. We first give a nondeterministic linear space machine. The machine starts with (s, F, v) and the characteristic vector of C written on its input tape. It begins by writing the start vector s on its worktape. If $s = C$, it halts and answers yes. Otherwise it guesses an $a \in \Sigma$ and overwrites its worktape contents with the characteristic vector of $\delta_F(s, a)$. If this is equal to C, it accepts, otherwise it guesses another letter and repeats. At any time, the machine must store only $O(n)$ bits of information. By Savitch's theorem, there is an equivalent deterministic machine running in $O(n^2)$ space. □

To construct p, the machine counts from 0 to $2^n - 1$ in binary (each number is identified with a subset of states of (s, F, v) by treating its binary representation as a characteristic vector). For each i between 0 and $2^n - 1$, it tests whether i represents an accessible state. If i does not, the machine proceeds to the next i. If i does represent an accessible state, the machine outputs 1 if i represents precisely the set of start states of (s, F, v), and 0 otherwise. The construction of t is similar, except the machine outputs 1 if any of the states in the subset represented by i are accept states, and 0 if none are.

The construction of D, the transition matrix, requires three counters. The first two, i and j, range from 0 to $2^n - 1$, and are used to keep track of the rows and columns of D, respectively. The

third counter, c, ranges from 0 to $m-1$, where $m = |\Sigma|$. The machine starts with all counters set to zero. It begins by testing i for accessibility. If i is inaccessible, it increments i and repeats. If i does correspond to an accessible state, it then tests each possible value of j for accessibility. If j is not accessible, it increments j. If j does represent an accessible state, it tests each $a_k \in \Sigma$ to determine whether $\delta'_F(i, a_k) = j$. If yes, it outputs a_k. If none of the a_k tests succeed, it outputs 0. After testing all of the a_i's, the machine resets c to 0 and goes to the next j. After checking all of the j's, the machine resets j to 0 and goes to the next i.

This term-generating transducer runs in $O(n^2)$ space, where n is the size of (s, F, v). The machine requires $O(n^2)$ space to perform the test in lemma 2 and a few counters which range up to $2^n - 1$. The alphabet is fixed, so we do not need to worry about $|\Sigma|$.

Let d be the size of (p, D, t) and n be the size of (s, F, v). Let X be the $d \times n$ matrix encoding the relation in which a state of (p, D, t) is related to all of the states of (s, F, v) that it "contains". The relation denoted by X makes the diagram in lemma 1 commute, so the equation $XF = DX$ holds. The equations $p^{\mathrm{T}}X = s^{\mathrm{T}}$ and $Xv = t$ are also easily seen to be true. Therefore the proof-generating machine can use the bisimulation lemma to prove the equivalence of (s, F, v) and (p, D, t). In this case, (p, D, t) is the source automaton.

We must show that the bisimulation matrix can be computed without violating the space bound. The term-generating transducer which takes the pair $((s, F, v), (p, D, t))$ and outputs the bisimulation matrix can use only polynomially many (in $|(s, F, v)|$) cells, although (p, D, t) may be exponential in n. To construct X, the machine needs one counter ranging from 0 to $2^n - 1$. For each i between 0 and $2^n - 1$, the machine tests the subset of states encoded by i for accessibility. If it is accessible, it outputs the binary representation of i as a row with n entries. If i does not represent an accessible state, it goes to $i + 1$.

The equations are true and the terms can be constructed; we now describe the $LOGSPACE$ proof-generating transducer which takes as input the triple $((s, F, v), (p, D, t), X)$ and outputs proofs of

$$p^{\mathrm{T}}X = s^{\mathrm{T}}$$

$$Xv = t$$

$$XF = DX.$$

The proof that $p^{\mathrm{T}}X = s^{\mathrm{T}}$ is actually n many proofs, each one involving the dot product of two vectors of size d. However, these are simple equations involving sums of four possible products: 00, 01, 10, and 11. The machine scans through the dot product, replacing each product with the appropriate element of $\{0, 1\}$ in turn. It then simplifies the expression using the facts that $0 + 0 = 0$, $1 + 0 = 1$, $0 + 1 = 1$, and $1 + 1 = 1$. For example, to prove

$$00 + 10 + 01 + 11 = 1,$$

the machine outputs:

$$\begin{aligned} 00 + 10 + 01 + 11 &= 0 + 10 + 01 + 11 \\ &= 0 + 0 + 10 + 11 \\ &= 0 + 0 + 0 + 11 \\ &= 0 + 0 + 0 + 1 \\ &= 0 + 0 + 1 \\ &= 0 + 1 \\ &= 1. \end{aligned}$$

This can be done in $LOGSPACE$ because the machine does not need to remember any intermediate terms. If the machine always simplifies from left to right and remembers how many times it

has performed each type of simplification, it can generate the next equation from the dot product, which is essentially stored on its input tape. The proof that $Xv = t$ is similar; there are d many equations, each involving a dot product of two vectors of size n.

The proof that $XF = DX$ is more complicated, since the terms involved contain letters of Σ, and not just 0 and 1. For each i, j, the machine must output a proof that $XF_{ij} = DX_{ij}$. The strategy is to define a normal form on XF_{ij} and DX_{ij} and prove that each term is equivalent to its normal form. The normal form is:

$$(\cdots((a_{k_1} + a_{k_2}) + a_{k_3}) + a_{k_4}) + \cdots) + a_{k_m})$$

with $a_{k_r} \in \Sigma$ and the ordering on the a_k's determined by an arbitrary but fixed order on Σ.

To prove that DX_{ij} is equivalent to its normal form, the machine first scans the i^{th} row of D and the j^{th} column of X to output their dot product, without doing any algebraic simplifications. Now, since (p, D, t) is deterministic, at most $|\Sigma|$ many entries of the i^{th} row of D are nonzero. By repeatedly simplifying using the axioms involving 0, the machine can output smaller and smaller equivalent terms. Eventually, the size of the term is approximately $|\Sigma|$, and the machine has enough space to put it in its normal form using the idempotent semiring axioms.

To prove that XF_{ij} is equivalent to its normal form, the machine scans through the i^{th} row of X and the j^{th} column of F and outputs their dot product. If n is the size of (s, F, v), then the length of this dot product is $O(n)$, and the machine can put the dot product into its normal form by again applying the idempotent semiring axioms.

The transducer P uses these procedures to generate (p_1, D_A, t_1), a deterministic automaton accepting α, and (p_2, D_B, t_2), a deterministic automaton accepting β. It does not have enough room to store the descriptions of these automata, but it can use the procedure described in lemma 2 to generate proofs that $s_1^{\mathrm{T}} F_A^* t_1 = p_1^{\mathrm{T}} D_A^* t_1$ and $s_2^{\mathrm{T}} F_B^* t_2 = p_2^{\mathrm{T}} D_B^* t_2$ without violating the space bound. The machine then outputs $\alpha = p_1^{\mathrm{T}} D_A^* t_1$ and $\beta = p_2^{\mathrm{T}} D_B^* t_2$, which follow from what is already on the output tape by transitivity.

4.4 Deterministic Automaton to Minimal Deterministic Automaton

At this stage, we require two term-generating transducers. The first constructs the minimal deterministic automaton equivalent to a given deterministic automaton, and the second takes as input a pair (dfa, equivalent minimal dfa) and outputs the bisimulation matrix between them. The minimal dfa (q, M, r) is constructed by examining (p, D, t) and outputting the least-numbered state in each equivalence class of a Myhill-Nerode relation. Recall that two states of (p, D, t), i and j, are equivalent (indistinguishable) if and only if for all $w \in \Sigma^*$, $\delta_D(i, w)$ and $\delta_D(j, w)$ are either both accept states or both nonaccept states. We require a lemma establishing a space bound on the procedure to identify equivalent states.

Lemma 4. *Let (p, D, t) be a deterministic automaton. It is decidable in polylog space whether i and j, two states of (p, D, t), are equivalent.*

Proof. We first give an $NLOGSPACE$ procedure to recognize distinguishable states. The machine begins with (p, D, t), i, and j written on its input tape. If one of i, j is an accept state and the other is not, the machine halts and answers distinguishable. Otherwise it guesses an $a_1 \in \Sigma$ and overwrites its worktape contents with $\delta_D(i, a_1)$ and $\delta_D(j, a_1)$. If exactly one of these states is an accept state, the machine halts and answers distinguishable. If not, it guesses an $a_2 \in \Sigma$ and repeats the procedure. At any time, the machine has to remember only two states of (p, D, t), and so it runs in $NLOGSPACE$. By Savitch's theorem, there is an equivalent deterministic machine running in $O((\log |(p, D, t)|)^2)$ space. □

To construct q, the start vector, the machine scans p. For each state i, it checks whether i is equivalent to some lower-numbered state. If yes, it skips to the next i. If i is the least-numbered

state in its equivalence class, the machine outputs a 1 if i is equivalent to the start state of (p, D, t), and 0 otherwise. The accept vector, r, is constructed similarly. The machine scans through t, and for each state i that is the least-numbered state in its equivalence class, it outputs 1 if i is an accept state, 0 if i is not.

The construction of the transition matrix M resembles the construction of the transition matrix of the deterministic automaton in stage 4.3. The machine maintains two counters, i and j. It scans through the states of (p, D, t), and for each state i which is the least-numbered state in its equivalence class, it tests each state j in turn, outputting D_{ij} for each j which is the first state in its equivalence class. It is easy to see that this procedure can be done in $PLSPACE$ and does indeed generate the equivalent minimal dfa.

These automata are proven equivalent using the bisimulation lemma. We must show that there is a matrix satisfying the hypotheses of lemma 1, and that this matrix can be computed within the space bound. Let

$$R \subseteq \mathcal{D} \times \mathcal{M}$$

such that $(i, j) \in R$ if and only if state i of (p, D, t) and state j of (q, M, r) are indistinguishable. That is, if there is no word w such that starting at state i in (p, D, t) and processing w leads to an accept state, whereas starting at state j in (q, M, r) and processing w leads to a fail state, or vice versa. It is clear that R makes the diagram in lemma 1 commute, and so $XM = DX$, where X is the realization of R as a 0-1 matrix. The source automaton is again (p, D, t).

We must now prove $p^{\mathrm{T}}X = q^{\mathrm{T}}$ holds. Since (p, D, t) and (q, M, r) are equivalent, the start state of (p, D, t) is related to the start state of (q, M, r), so $p^{\mathrm{T}}X$ has a 1 in the entry corresponding to the start state of (q, M, r). To see that the other entries of $p^{\mathrm{T}}X$ are 0, note that each state of (p, D, t) is related to exactly one state of (q, M, r), by minimality of (q, M, r). A 1 in an entry of $p^{\mathrm{T}}X$ not corresponding to the start state of (q, M, r) would mean that there is another state of (q, M, r) which is indistinguishable from the start state of (p, D, t), and therefore indistinguishable from the start state of (q, M, r), contradicting the minimality of (q, M, r). Note that this is a non-trivial use of minimality and partly explains the fact that even though any two equivalent deterministic automata are bisimilar via R (in the standard, non-algebraic sense), we cannot necessarily use the bisimulation lemma to prove their equivalence if neither are minimal.

Finally, we show $Xr = t$. Let s_M be the start state of (q, M, r) and s_D be the start state of (p, D, t). Every state in (p, D, t) is accessible, so for any accept state i of (p, D, t), there is a word w such that $\hat{\delta}_D(s_D, w) = i$. Since (q, M, r) is deterministic and equivalent to (p, D, t), the state $\hat{\delta}_M(s_M, w)$ must be an accept state and related to i. No nonaccept state of (p, D, t) can be related to an accept state of (q, M, r), by the definition of R. These considerations imply $Xr = t$.

The hypotheses of the bisimulation lemma are satisfied, so it can be used to prove the equivalence of (p, D, t) and (q, M, r). A term-generating transducer to construct X from the pair $((p, D, t), (q, M, r))$ uses a straightforward modification of lemma 4 to generate X in $PLSPACE$.

The proof-generating transducer which takes in the triple $((p, D, t), (q, M, r), X)$ to prove the equivalence of the two automata applies a straightforward modification of the procedure used in stage 4.3. Note that both automata may be exponential in n, but lemma 2 ensures that everything can be done in $PSPACE$.

The transducer P uses these procedures to generate (q_1, M_A, r_1), the minimal deterministic automaton accepting α, and a proof that $p_1^{\mathrm{T}} D_A^* t_1 = q_1^{\mathrm{T}} M_A^* r_1$. It then outputs $\alpha = q_1^{\mathrm{T}} M_A^* r_1$, which follows by transitivity.

4.5 DFA for β Equivalent to Minimal Automaton for α

At this stage, P has proven $\alpha = q_1^{\mathrm{T}} M_A^* r_1$ and $\beta = p_2^{\mathrm{T}} D_B^* t_2$. It remains for P to prove that $q_1^{\mathrm{T}} M_A^* r_1 = p_2^{\mathrm{T}} D_B^* t_2$. It suffices for P to use the term-generating procedure from the previous stage with input $((p_2, D_B, t_2), (q_1, M_A, r_1))$ to generate the bisimulation matrix between the two

automata, and then to use the proof-generating procedure from stage 4.4 to output a proof that $q_1^{\mathrm{T}} M_A^* r_1 = p_2^{\mathrm{T}} D_B^* t_2$. It can then output $\beta = q_1^{\mathrm{T}} M_A^* r_1$ and finally $\alpha = \beta$, both of which follow by transitivity of equality.

5 Proving KAT Equations

We now shift our focus to KAT. Our main theorem is the following.

Theorem 2. *Let t_1 and t_2 be two equivalent KAT terms. A proof that $t_1 = t_2$ can be produced by a PSPACE transducer.*

Theorem 2 is proven by efficiently reducing KAT equations to KA equations, then applying the algorithm in section 4.

We first provide an overview of *guarded string algebras*, which are models of the KAT axioms. For a more detailed introduction, see [5]. Guarded string algebras play the same role for KAT that regular languages do for KA; two KAT terms t_1 and t_2 are equivalent modulo the axioms of Kleene algebra with tests if and only if they denote the same set of guarded strings.

Let P and B be finite alphabets. Elements of P are called atomic programs, and elements of B are called atomic tests. Guarded strings are obtained from each word $w \in P^*$ by interspersing atoms of the free Boolean algebra on B among the letters of w (we require that a guarded string both begins and ends with an atom). Let $b_1, b_2, ..., b_n$ be the elements of B. Recall that an atom α of the free Boolean algebra on B is a product of the form

$$\alpha = c_1 c_2 \cdots c_n$$

where $c_i \in \{b_i, \overline{b_i}\}$ for each i. We require an ordering on the literals appearing in an atom so that there is a unique string denoting each atom. Let A_B denote the set of atoms.

Given a guarded string x, let first(x) be the leftmost atom of x, and last(x) be the rightmost atom of x. We define a partial concatenation operation on guarded strings, denoted $\diamond$, as follows. Given two guarded strings, x and y, let $x = x'\alpha$ and $y = \beta y'$, where $\alpha =$last(x) and $\beta =$ first(y).

Define

$$x \diamond y = x'\alpha y', \text{ if } \alpha = \beta, \text{ undefined otherwise.}$$

We now give interpretations of the KAT operations on sets of guarded strings. Let C and D be sets of guarded strings. Define

$$\begin{aligned} &C + D = C \cup D \\ &C \cdot D = \{x \diamond y \mid x \in C,\ y \in D\} \\ &C^0 = A_B \\ &C^* = \textstyle\bigcup_{n \geq 0}\ C^n. \end{aligned}$$

We must also interpret the complementation function. Let C be a set of guarded strings such that $C \subseteq A_B$. Define

$$\overline{C} = A_B - C.$$

Using these operations, we can define a function G from KAT terms to sets of guarded strings inductively. The base cases are:

$$\begin{aligned} &G(0) = \emptyset \\ &G(1) = \{\alpha \mid \alpha \in A_B\} \\ &G(b) = \{\alpha \mid \alpha \to b \text{ is a propositional tautology}\} \\ &G(p) = \{\alpha p \beta \mid \alpha, \beta \in A_B\}. \end{aligned}$$

In [5], the completeness of the guarded string model for the equational theory of KAT is shown by a reduction from the equational theory of KAT to the equational theory of KA. This is achieved

by transforming a KAT term t into a KAT-equivalent term t' such that $R(t') = G(t)$. Unfortunately, the term t' may be exponentially longer than t, which might make the proof constructed in section 4 doubly exponential (determinizing could cause another exponential blowup). We give an alternate construction. Given a term t, we construct an automaton (u, A, v) such that $t = u^{\mathrm{T}} A^* v$ modulo the axioms of KAT, and (u, A, v) accepts precisely the set of guarded strings denoted by t. The automaton (u, A, v) will be polynomial in the size of t.

We need a few additional theorems of Kleene algebra in our construction. As in section 3, the hypotheses are quite simple, so proofs using these theorems are efficiently verifiable.

5.1 More Theorems of KA

The extra axioms satisfied by Boolean terms, particularly multiplicative idempotence and star-triviality, complicate the construction of the automaton. We overcome these difficulties by selectively applying the Boolean axioms to Boolean terms. That is, we first treat Boolean terms simply as words over an alphabet, and apply the lemmas below. However, these lemmas produce automata which are not simple. In the inductive construction in section 5.2, we then use the Boolean axioms to simplify the transition matrices. Note, however, that the two lemmas below are theorems of Kleene algebra, and do not require the Boolean axioms.

5.1.1 The Second Concatenation Lemma

The *second concatenation* lemma is based on the following alternate way of constructing an automaton accepting the concatenation of two languages. In section 3, we encoded the standard construction of such an automaton by connecting the accept states of the first automaton to the start states of the second with ϵ-transitions. However, we could also do the following: for each state i of (u, A, v) with an outgoing x transition to an accept state, and each state j of (s, B, t) with an incoming y transition from a start state, add an xy transition from i to j. Note that we allow x and y to be arbitrary elements of a Kleene algebra, not just letters in Σ. This construction yields an automaton accepting $u^{\mathrm{T}} A^* v s^{\mathrm{T}} B^* t$, provided neither (u, A, v) nor (s, B, t) has a state which is both a start state and an accept state, which we can represent algebraically as $u^{\mathrm{T}} v = 0,\ s^{\mathrm{T}} t = 0$. This idea is the crux of the second concatenation lemma. The lemma itself looks rather complicated, so we explain how it will be used. In the construction in 5.2, we will have two ϵ-free automata, (u_1, A_1, v_1) and (u_2, A_2, v_2). Each of these automata will be the disjoint union of two automata:

$$(u_i, A_i, v_i) = \left(\left[\begin{array}{c} o_i \\ \hline s_i \end{array}\right], \left[\begin{array}{c|c} C_i & 0 \\ \hline 0 & B_i \end{array}\right], \left[\begin{array}{c} r_i \\ \hline t_i \end{array}\right] \right).$$

It will be the case that neither of them accept the empty word, i.e.,

$$o_i^{\mathrm{T}} r_i = 0$$

$$s_i^{\mathrm{T}} t_i = 0$$

for $i = 1, 2$. The construction will require an automaton accepting

$$L = (o_1^{\mathrm{T}} C_1^* r_1 s_2^{\mathrm{T}} B_2^* t_2) + (s_1^{\mathrm{T}} B_1^* t_1 o_2^{\mathrm{T}} C_2^* r_2) + (s_1^{\mathrm{T}} B_1^* t_1 s_2^{\mathrm{T}} B_2^* t_2).$$

The second concatenation lemma,

$$\frac{\Phi \vdash o_1^{\mathrm{T}} r_1 = 0 \qquad \Phi \vdash o_2^{\mathrm{T}} r_2 = 0 \qquad \Phi \vdash s_1^{\mathrm{T}} t_1 = 0 \qquad \Phi \vdash s_2^{\mathrm{T}} t_2 = 0}{\Phi \vdash \left[\begin{array}{c} o_1 \\ s_1 \\ 0 \\ 0 \end{array}\right]^{\mathrm{T}} \left[\begin{array}{cc|cc} C_1 & 0 & 0 & C_1 r_1 s_2^{\mathrm{T}} B_2 \\ 0 & B_1 & B_1 t_1 o_2^{\mathrm{T}} C_2 & B_1 t_1 s_2^{\mathrm{T}} B_2 \\ \hline 0 & 0 & C_2 & 0 \\ 0 & 0 & 0 & B_2 \end{array}\right]^* \left[\begin{array}{c} 0 \\ 0 \\ r_2 \\ t_2 \end{array}\right] = L}$$

allows us to do this.

The proof is a straightforward calculation:

$$\begin{bmatrix} o_1 \\ s_1 \\ 0 \\ 0 \end{bmatrix}^{\mathrm{T}} \left[\begin{array}{cc|cc} C_1 & 0 & 0 & C_1 r_1 s_2^{\mathrm{T}} B_2 \\ 0 & B_1 & B_1 t_1 o_2^{\mathrm{T}} C_2 & B_1 t_1 s_2^{\mathrm{T}} B_2 \\ \hline 0 & 0 & C_2 & 0 \\ 0 & 0 & 0 & B_2 \end{array}\right]^* \begin{bmatrix} 0 \\ 0 \\ r_2 \\ t_2 \end{bmatrix} =$$

$$o_1^{\mathrm{T}} C_1^* C_1 r_1 s_2^{\mathrm{T}} B_2 B_2^* t_2 + s_1^{\mathrm{T}} B_1^* B_1 t_1 o_2^{\mathrm{T}} C_2 C_2^* r_2 + s_1^{\mathrm{T}} B_1^* B_1 t_1 s_2^{\mathrm{T}} B_2 B_2^* t_2.$$

Using the hypotheses, it is easy to show that this sum is equal to L. The proofs involved are of the following form:

$$\begin{aligned} o_1^{\mathrm{T}} C_1^* r_1 &= o_1^{\mathrm{T}}(1 + C_1^* C_1) r_1 \\ &= o_1^{\mathrm{T}} r_1 + o_1^{\mathrm{T}} C_1^* C_1 r_1 \\ &= o_1^{\mathrm{T}} C_1^* C_1 r_1. \end{aligned}$$

5.1.2 The Second Asterate Lemma

Let (u, A, v) be a simple, ϵ-free automaton and γ be a regular expression. Suppose $u^{\mathrm{T}} A^* v = \gamma$. The standard construction of an automaton accepting $\gamma\gamma^*$ proceeds by adding ϵ-transitions from the accept states of (u, A, v) back to its start states. Suppose (u, A, v) has no paths of length 0 or 1 from a start state to an accept state, which we can model algebraically as $u^{\mathrm{T}} v = 0, u^{\mathrm{T}} A v = 0$. In this case, we can construct an automaton accepting $\gamma\gamma^*$ from (u, A, v) with the following procedure: for each state i with an outgoing x transition to an accept state, and each state j with an incoming y transition from a start state, add an xy transition from i to j. This automaton, although not simple, accepts $\gamma\gamma^*$. This idea is the basis of the *second asterate* lemma.

Suppose (u, A, v) is the disjoint union of two automata, (o, C, r) and (s, B, t). Also suppose that $o^{\mathrm{T}} C^* r \leq 1$, and $s^{\mathrm{T}} t + s^{\mathrm{T}} B t = 0$, which implies $s^{\mathrm{T}} B^* t = s^{\mathrm{T}} B^* B B t$. Under these conditions, we can apply the second asterate lemma:

$$\frac{\Phi \vdash o^{\mathrm{T}} C^* r \leq 1 \qquad \Phi \vdash s^{\mathrm{T}} B^* t = s^{\mathrm{T}} B^* B B t}{\Phi \vdash \left(\begin{bmatrix} o \\ \hline s \end{bmatrix}^{\mathrm{T}} \left[\begin{array}{c|c} C & 0 \\ \hline 0 & B \end{array}\right]^* \begin{bmatrix} r \\ \hline t \end{bmatrix}\right)^* = \begin{bmatrix} 1 \\ \hline s \end{bmatrix}^{\mathrm{T}} \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & B + B t s^{\mathrm{T}} B \end{array}\right]^* \begin{bmatrix} 1 \\ \hline t \end{bmatrix}}.$$

Note that $B + Bts^{\mathrm{T}}B$ algebraically encodes the alternate asterate construction.

Since (u, A, v) is the disjoint union of (o, C, r) and (s, B, t), it is easy to show (cf. section 3.1) that

$$u^{\mathrm{T}} A^* v = o^{\mathrm{T}} C^* r + s^{\mathrm{T}} B^* t.$$

By KA axiom 10,

$$(u^{\mathrm{T}} A^* v)^* = 1 + u^{\mathrm{T}} A^* v (u^{\mathrm{T}} A^* v)^*.$$

We can now substitute:

$$1 + u^{\mathrm{T}} A^* v (u^{\mathrm{T}} A^* v)^* = 1 + (o^{\mathrm{T}} C^* r + s^{\mathrm{T}} B^* t)(o^{\mathrm{T}} C^* r + s^{\mathrm{T}} B^* t)^*.$$

By the denesting rule of Kleene algebra,

$$1 + (o^{\mathrm{T}} C^* r + s^{\mathrm{T}} B^* t)(o^{\mathrm{T}} C^* r + s^{\mathrm{T}} B^* t)^* = 1 + (o^{\mathrm{T}} C^* r + s^{\mathrm{T}} B^* t)(o^{\mathrm{T}} C^* r)^* (s^{\mathrm{T}} B^* t (o^{\mathrm{T}} C^* r)^*)^*.$$

Since $o^{\mathrm{T}} C^* r \leq 1, (o^{\mathrm{T}} C^* r)^* = 1$. We can simplify:

$$1 + (o^{\mathrm{T}}C^*r + s^{\mathrm{T}}B^*t)(o^{\mathrm{T}}C^*r)^*(s^{\mathrm{T}}B^*t(o^{\mathrm{T}}C^*r)^*)^* = 1 + (o^{\mathrm{T}}C^*r + s^{\mathrm{T}}B^*t)(s^{\mathrm{T}}B^*t)^*.$$

By distributivity and axiom 10 again,

$$1 + (o^{\mathrm{T}}C^*r + s^{\mathrm{T}}B^*t)(s^{\mathrm{T}}B^*t)^* = 1 + s^{\mathrm{T}}B^*t(s^{\mathrm{T}}B^*t)^*.$$

At this point, we have shown that $u^{\mathrm{T}}A^*v = 1 + s^{\mathrm{T}}B^*t(s^{\mathrm{T}}B^*t)^*$. It remains to be shown that under the assumption $s^{\mathrm{T}}B^*t = s^{\mathrm{T}}B^*BBt$,

$$s^{\mathrm{T}}B^*t(s^{\mathrm{T}}B^*t)^* = s^{\mathrm{T}}(B + Bts^{\mathrm{T}}B)^*t. \qquad (1)$$

The first part of the proof is similar to the proof in section 3.3.

$$\begin{aligned} s^{\mathrm{T}}B^*t(s^{\mathrm{T}}B^*t)^* &= s^{\mathrm{T}}B^*BBt(s^{\mathrm{T}}B^*BBt)^* \\ &= s^{\mathrm{T}}B^*B(Bts^{\mathrm{T}}B^*B)^*Bt \\ &= s^{\mathrm{T}}BB^*(Bts^{\mathrm{T}}BB^*)^*Bt \\ &= s^{\mathrm{T}}B(B + Bts^{\mathrm{T}}B)^*Bt. \end{aligned}$$

The following equation is an easy consequence of the axioms of Kleene algebra:

$$(B + Bts^{\mathrm{T}}B)^* = 1 + Bts^{\mathrm{T}}B(B + Bts^{\mathrm{T}}B)^* + (B + Bts^{\mathrm{T}}B)^*Bts^{\mathrm{T}}B + B(B + Bts^{\mathrm{T}}B)^*B.$$

Multiplying the equation on the left by s^{T}, on the right by t, and simplifying using $s^{\mathrm{T}}t = 0$ and $s^{\mathrm{T}}Bt = 0$ yields

$$s^{\mathrm{T}}(B + Bts^{\mathrm{T}}B)^*t = s^{\mathrm{T}}B(B + Bts^{\mathrm{T}}B)^*Bt.$$

This proves (1). We now add the trivial one-state automaton to the automaton $(s, B + Bts^{\mathrm{T}}B, t)$, completing the proof of the second asterate lemma.

5.2 KAT Term to Automaton

In this section, we give the transducer which takes as input a KAT term t and outputs an automaton accepting $G(t)$. This transducer is both term-generating and proof-generating. Before constructing the automaton, it must convert t into a well-behaved form.

5.2.1 Only Complement Primitive Tests

The machine first uses the De Morgan laws and the Boolean axiom $\overline{\overline{b}} = b$ to transform a term t into an equivalent term t' in which the complementation symbol is only applied to atomic tests. If we interpret t' as a regular expression, then $R(t') \subseteq (P \cup B \cup \overline{B})^*$, where $\overline{B} = \{\overline{b} \mid b \in B\}$. The transducer works as follows. On input t, it copies t onto its worktape and onto the output tape. Then, starting at the root of the syntax tree of t, it works it way down the tree until it finds a subtree containing only Boolean terms such that either some term is complemented twice, or a conjunction or disjunction appears under the complement symbol. It then applies the appropriate axiom to this subtree, overwrites its worktape contents, and then outputs the updated term. The machine then begins searching again at the root of the tree. When it scans the whole tree and does not have to apply any axioms, it stops. The transducer requires only polynomially many worktape cells. Furthermore, the increase in the size of the term is negligible. At the end of this stage, it has t' written on its worktape.

5.2.2 New variables for atoms

For the remainder of the construction, it is advantageous to treat each atom as a single letter. Let $z = 2^{|B|}$. The machine generates z many new variables, $x_1, x_2, ..., x_z$. For each i, it outputs the equation

$$x_i = \alpha_i$$

where α_i is the i^{th} atom. The automaton constructed below uses the alphabet $P \cup \{x_1, x_2, ..., x_z\}$. It is a routine matter to verify that two KAT terms denote same set of guarded strings if and only if they denote the same set of words after performing this substitution. For the rest of the construction, we use the terms "guarded strings" and "guarded strings after this substitution" interchangeably.

5.2.3 Constructing the Automaton

Now that the preprocessing of the term is complete, the machine constructs the automaton. The construction is inductive and resembles the construction in 4.1. However, the machine will maintain several invariants throughout the construction which were not necessary in the pure Kleene algebra case. At a given substage, let (u, A, v) be the final automaton constructed. The automaton (u, A, v) will satisfy:

- (u, A, v) is simple and ϵ-free.
- (u, A, v) is the disjoint union of two (possibly empty) automata, (o, C, r) and (s, B, t).
- (s, B, t) accepts only words of length two or more, so., $s^{\mathrm{T}} B^* t = s^{\mathrm{T}} B^* BBt$.
- (o, C, r) is a two state automaton accepting only one-letter words from the alphabet $\{x_1, x_2, ..., x_z\}$.
- The first two states of (u, A, v) are the states of (o, C, r) (if (o, C, r) is nonempty).

The base case of the induction is as follows. For an atomic term a, $\hat{a}$ denotes the automaton constructed.

$$\hat{0} = (0, 0, 0)$$

$$\hat{1} = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & \sum_i x_i \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right)$$

$$\hat{b} = \left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & \sum_{x_i \leq b} x_i \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right)$$

$$\hat{p} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & \sum_i x_i & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 0 & 0 & \sum_i x_i \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right)$$

For each automaton, the machine must prove that the language it accepts is KAT-equivalent to the appropriate atomic term. There are finitely many atomic terms, so the machine can store all of the necessary proofs in its finite control. Note that this expansion increases the size of a term by only a constant amount, although the constant is exponential in $|B|$. Cf. the proof that the Boolean algebra axioms entail all propositional tautologies.

We now treat the inductive step of the construction. The easiest automaton to construct is that for addition. Suppose we have two automata (u_1, A_1, v_1) and (u_2, A_2, v_2), such that $u_1^{\mathrm{T}} A_1^* v_1 = \gamma$ and $u_2^{\mathrm{T}} A_2^* v_2 = \delta$. By induction, (u_1, A_1, v_1) is the disjoint union of (o_1, C_1, r_1) and (s_1, B_1, t_1), and (u_2, A_2, v_2) is the disjoint union of (o_2, C_2, r_2) and (s_2, B_2, t_2). The machine first proves the equations

$$u_1^{\mathrm{T}} A_1^* v_1 = o_1^{\mathrm{T}} C_1^* r_1 + s_1^{\mathrm{T}} B_1^* t_1$$
$$u_2^{\mathrm{T}} A_2^* v_2 = o_2^{\mathrm{T}} C_2^* r_2 + s_2^{\mathrm{T}} B_2^* t_2.$$

It then outputs a proof that

$$\gamma + \delta = (o_1^{\mathrm{T}} C_1^* r_1 + o_2^{\mathrm{T}} C_2^* r_2) + s_1^{\mathrm{T}} B_1^* t_1 + s_2^{\mathrm{T}} B_2^* t_2.$$

The machine can now construct a two-state automaton (o, C, r) which accepts $(o_1^{\mathrm{T}} C_1^* r_1 + o_2^{\mathrm{T}} C_2^* r_2)$, then apply the addition construction from 4.1 to $(o, C, r), (s_1, B_1, t_1)$, and (s_2, B_2, t_2). This yields an automaton (u, A, v) which satisfies the invariants and accepts $\gamma + \delta$. Note that there are only finitely many possibilities for (o_1, C_1, r_1) and (o_2, C_2, r_2), so the machine can prove

$$o^{\mathrm{T}} C^* r = o_1^{\mathrm{T}} C_1^* r_1 + o_2^{\mathrm{T}} C_2^* r_2$$

using data from its finite control.

The automaton for the product of two terms is more complicated. Again, let (u_1, A_1, v_1) and (u_2, A_2, v_2) be two automata such that $u_1^{\mathrm{T}} A_1^* v_1 = \gamma$ and $u_2^{\mathrm{T}} A_2^* v_2 = \delta$. As in the case for addition, we use the fact that each of these automata is the disjoint union of two automata:

$$u_1^{\mathrm{T}} A_1^* v_1 = o_1^{\mathrm{T}} C_1^* r_1 + s_1^{\mathrm{T}} B_1^* t_1$$
$$u_2^{\mathrm{T}} A_2^* v_2 = o_2^{\mathrm{T}} C_2^* r_2 + s_2^{\mathrm{T}} B_2^* t_2.$$

The machine can output a proof of the equations

$$\gamma\delta = (o_1^{\mathrm{T}} C_1^* r_1 + s_1^{\mathrm{T}} B_1^* t_1)(o_2^{\mathrm{T}} C_2^* r_2 + s_2^{\mathrm{T}} B_2^* t_2)$$
$$= (o_1^{\mathrm{T}} C_1^* r_1 o_2^{\mathrm{T}} C_2^* r_2) + (o_1^{\mathrm{T}} C_1^* r_1 s_2^{\mathrm{T}} B_2^* t_2) + (s_1^{\mathrm{T}} B_1^* t_1 o_2^{\mathrm{T}} C_2^* r_2) + (s_1^{\mathrm{T}} B_1^* t_1 s_2^{\mathrm{T}} B_2^* t_2).$$

The term $(o_1^{\mathrm{T}} C_1^* r_1 o_2^{\mathrm{T}} C_2^* r_2)$ is a sum of atoms after simplifying using the Boolean axioms. The machine can construct a two-state automaton (o, C, r) accepting this sum. Since there are only finitely many choices for $o_1^{\mathrm{T}} C_1^* r_1$ and $o_2^{\mathrm{T}} C_2^* r_2$, all of the necessary proofs can be stored in the finite control of the machine.

Let (s, B, t) be the automaton

$$\left(\begin{bmatrix} o_1 \\ s_1 \\ 0 \\ 0 \end{bmatrix}, \left[\begin{array}{cc|cc} C_1 & 0 & 0 & C_1 r_1 s_2^{\mathrm{T}} B_2 \\ 0 & B_1 & B_1 t_1 o_2^{\mathrm{T}} C_2 & B_1 t_1 s_2^{\mathrm{T}} B_2 \\ \hline 0 & 0 & C_2 & 0 \\ 0 & 0 & 0 & B_2 \end{array}\right], \begin{bmatrix} 0 \\ 0 \\ r_2 \\ t_2 \end{bmatrix} \right).$$

The machine first outputs proofs of the hypotheses of the second concatenation lemma. It can then output

$$s^{\mathrm{T}} B^* t = (o_1^{\mathrm{T}} C_1^* r_1 s_2^{\mathrm{T}} B_2^* t_2) + (s_1^{\mathrm{T}} B_1^* t_1 o_2^{\mathrm{T}} C_2^* r_2) + (s_1^{\mathrm{T}} B_1^* t_1 s_2^{\mathrm{T}} B_2^* t_2),$$

which follows from the second concatenation lemma.

The machine now constructs a simple automaton (s, B', t) by simplifying the transition matrix for (s, B, t) using the Boolean axioms and outputs a proof of the equivalence of (s, B, t) and (s, B', t). It then adds the automata (o, C, r) and (s, B', t) together to get (u, A, v), and outputs a proof of the equation

$$u^{\mathrm{T}} A^* v = \gamma\delta.$$

Finally, we come to the construction for *. Let (u, A, v) be an automaton such that $u^{\mathrm{T}} A^* v = \gamma$. This automaton is the disjoint union of two automata, (o, C, r) and (s, B, t) such that (o, C, r) accepts

a sum of atoms and (s, B, t) accepts no words of length less than two. The machine first outputs proofs that

$$o^{\mathrm{T}} C^* r \leq 1$$
$$s^{\mathrm{T}} B^* B B t = s^{\mathrm{T}} B t.$$

These facts follow from the Boolean axioms and the equation $s^{\mathrm{T}} t + s^{\mathrm{T}} B t = 0$.

The machine can now output

$$\left[\begin{array}{c} 1 \\ \hline s \end{array}\right]^{\mathrm{T}} \left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & B + B t s^{\mathrm{T}} B \end{array}\right]^* \left[\begin{array}{c} 1 \\ \hline t \end{array}\right] = \gamma^*,$$

which follows from the second asterate lemma. Finally, the machine can apply the Boolean axioms to each entry of

$$\left[\begin{array}{c|c} 1 & 0 \\ \hline 0 & B + B t s^{\mathrm{T}} B \end{array}\right]$$

to produce an equivalent simple, ϵ-free transition matrix D (1 becomes the sum of all atoms). It can then output a proof of

$$\left[\begin{array}{c} 1 \\ \hline s \end{array}\right]^{\mathrm{T}} \mathrm{D}^* \left[\begin{array}{c} 1 \\ \hline t \end{array}\right] = \gamma^*.$$

The proof that the automaton constructed for a term t accepts precisely the guarded strings denoted by t is a straightforward induction.

5.3 Proving KAT equations

A transducer P to prove KAT equations operates as follows. On input (t_1, t_2), it runs the above procedure to generate two simple ϵ-free automata, one accepting the set of guarded strings denoted by t_1, the other accepting the set of guarded strings denoted by t_2. It then performs the algorithm in section 4, starting at 4.3.

6 Reducing the Hoare Theory of KA(T) to the Equational Theory of KA

Finally, we make the simple observation that the reductions in [2] and [5] don't significantly increase the size of the terms.

Theorem 3. *Proofs of equational implications in the Hoare Theory of KA(T) can be produced by a PSPACE transducer.*

Proof. Given an alphabet $\Sigma = \{a_1, a_2, ..., a_n\}$, let $u = a_1 + a_2 + \cdots + a_n$. In [2], it is shown that

$$s \equiv t \Leftrightarrow s + uru = t + uru$$

is a Kleene algebra congruence, therefore $(r = 0 \rightarrow p = q) \leftrightarrow (p + uru = q + uru)$. The same reduction works for KAT, as is show in [5] - in this case u is only defined to be the sum of all of the atomic programs, not the atomic tests. The transformation from $r = 0 \rightarrow p = q$ to $p + uru = q + uru$ involves only a constant increase in size. □

7 Acknowledgments

I would like to thank Dexter Kozen for many helpful comments and informative conversations. This material is based upon work supported by the National Science Foundation under Grant No. 0635028.

References

[1] Berghammer, R., Möller, B. and Struth, G. (Eds.) *Relational and Kleene-Algebraic Methods in Computer Science*, May 2003.

[2] Cohen, Ernie. Hypotheses in Kleene Algebra. Technical Report TM-ARH-023814, Bellcore, 1993. http://citeseer.nj.nec.com/1688.html

[3] Cohen, E and Kozen, D. and Frederick, S. The Complexity of Kleene Algebra with Tests. *Technical Report 96-1598, Computer Science Department, Cornell University.* July 1996.

[4] Kozen, D. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Infor. and Comput*, 110(2):366-390. May 1994.

[5] Kozen, D. and Smith, Frederick. Kleene Algebra with Tests: Completeness and Decidability. *Proc. 10th Int. Workshop Computer Science Logic (CSL' 96)* 224-259. 1996.

[6] Kozen, D. Kleene Algebra with Tests. *Transactions on Programming Languages and Systems* 19:427-443. May 1997.

[7] Kozen, D. Typed Kleene Algebra. *Technical Report 98-1669, Computer Science Department, Cornell University.* March 1998.

[8] Kozen, D. On Hoare Logic and Kleene Algebra with Tests. *Trans. Computational Logic*, 1(1):60-76, July 2000.

[9] Kozen, D. and Tiuryn, Jerzy. On the Completeness of Propositional Hoare Logic. *Information Sciences*, 139:187-195, 2001.

[10] Necula, George C. and Lee, Peter. Proof-Carrying Code. *Technical Report CMU-CS-96-165*, Carnegie Mellon University, November 1996.

[11] Necula, George C. and Lee, Peter. Safe Kernel Extensions Without Run-time Checking. *Proc. 2nd Symp. Operating System Design and Implementation*, October 1996.

[12] Selman, A. Completeness of Calculii for Axiomatically Defined Classes of Algebras. *Algebra Universalis*, 2:20-32, 1972.

[13] Stockmeyer, L.J. and Meyer, A.R., Word Problems Requiring Exponential Time. *Proc. 5th Symp. Theory of Computing*, 1-9. 1973.